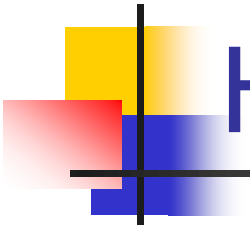
Hashing externo (II)

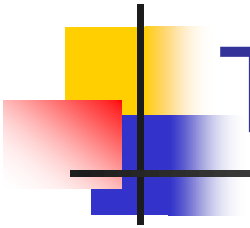
Graça Nunes

Fonte: Folk & Zoelick, File Structures



Hashing Extensível

- **Espalhamento Extensível** (*Extendible Hashing*): permite um auto-ajuste do espaço de endereçamento do espalhamento
  - Maior o número de chaves, maior o número de endereços
- Idéia chave é combinar o espalhamento convencional com uma técnica de recuperação de informações denominada **trie** (“trai”)



Trie

- também conhecida como *radix searching tree*, ou árvore de busca digital
- árvore de busca na qual o fator de sub-divisão, ou número máximo de filhos por nó, é igual ao número de símbolos do alfabeto que compõe as chaves
- boa opção para manter chaves grandes e de tamanho variável...
- origem do nome: palavra reTRIEval

Exemplo Tries

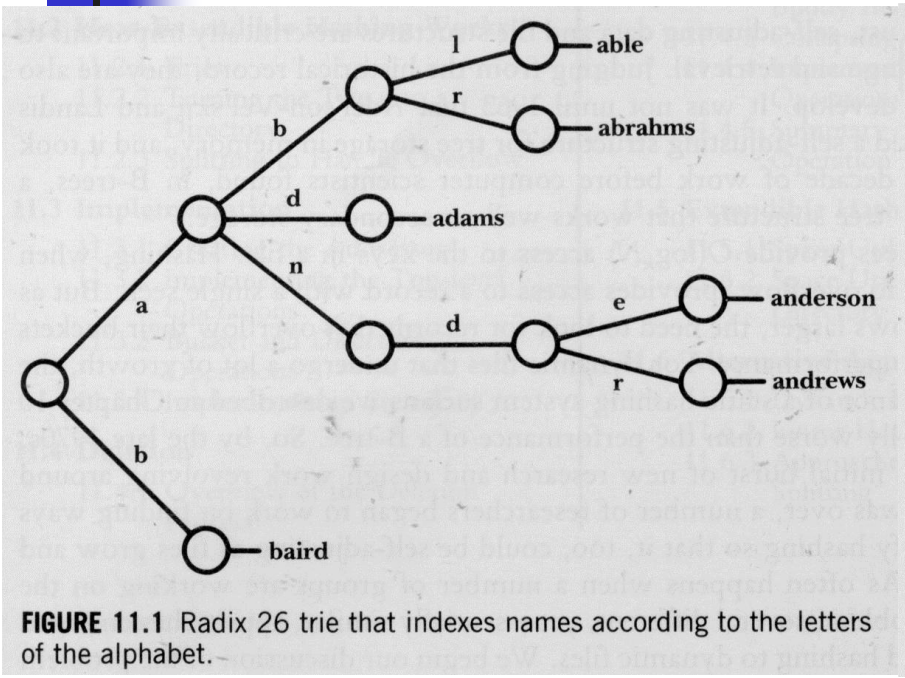


FIGURE 11.1 Radix 26 trie that indexes names according to the letters of the alphabet.

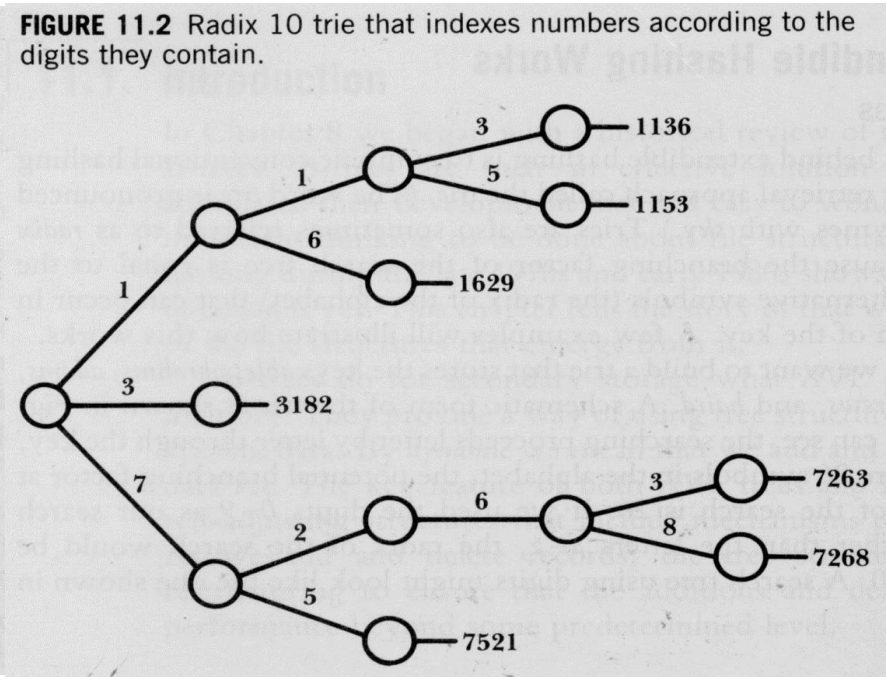
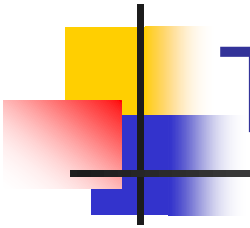


FIGURE 11.2 Radix 10 trie that indexes numbers according to the digits they contain.

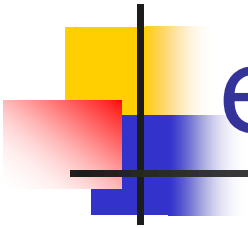
- Trie de ordem 26 (# letras)
- Trie de ordem 10 (# dígitos)



Tries

- Tempo de busca proporcional ao tamanho das chaves
- Chaves com sufixos comuns compartilham caminho até a raiz
- Propícias para compactação no caso de letras do alfabeto

Tries e espalhamento extensível



- Espalhamento extensível: tries de ordem 2
- Tabela de espalhamento indexa um conjunto de cestos (*buckets*)
- Função Hash gera um endereço binário
- Conjuntos de chaves (ou registros) são armazenados em cestos
- Busca por chave: análise bit-a-bit do valor de *h(key)* permite localizar o seu cesto

Tries e espalhamento extensível

- **Níveis internos:** rotulados com bits
- **Nível dos nós folha:** buckets contendo várias chaves ou registros
- Ex.: no bucket A, há chaves de endereço começando com 0; em B, endereços começando com 10; em C, com 11 (não foi possível colocar todas chaves de endereços começando com 1 num único bucket).

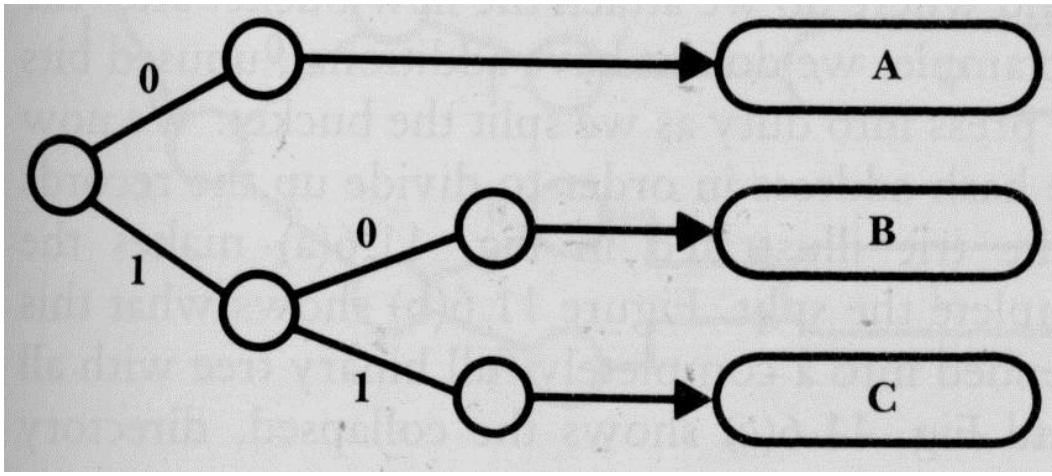
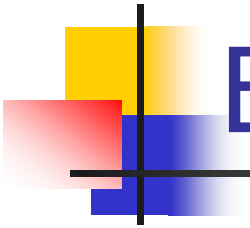
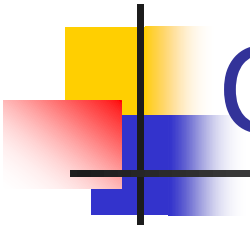


FIGURE 11.3 Radix 2 trie that provides an index to buckets.



Bucket (Cesto)

- Segmento físico útil de armazenamento externo
  - página, trilha ou segmento de trilha
- Análogo aos blocos da Árvore-B^+, porém não são ordenados.



Como representar a *trie*?

- Se for mantida como uma árvore, são necessárias várias comparações para descer ao longo de sua estrutura
  - Muitas comparações se forem necessários vários bits para diferenciar os buckets
  - Se o índice tb estiver em disco, esse caminho pode representar vários seeks adicionais
- Solução mais eficiente: representá-la como um diretório de endereços
  - Ou seja, linearizamos a estrutura

Transformando uma trie em um diretório

- A trie deve ser uma árvore binária completa
- Se não for, pode ser extendida

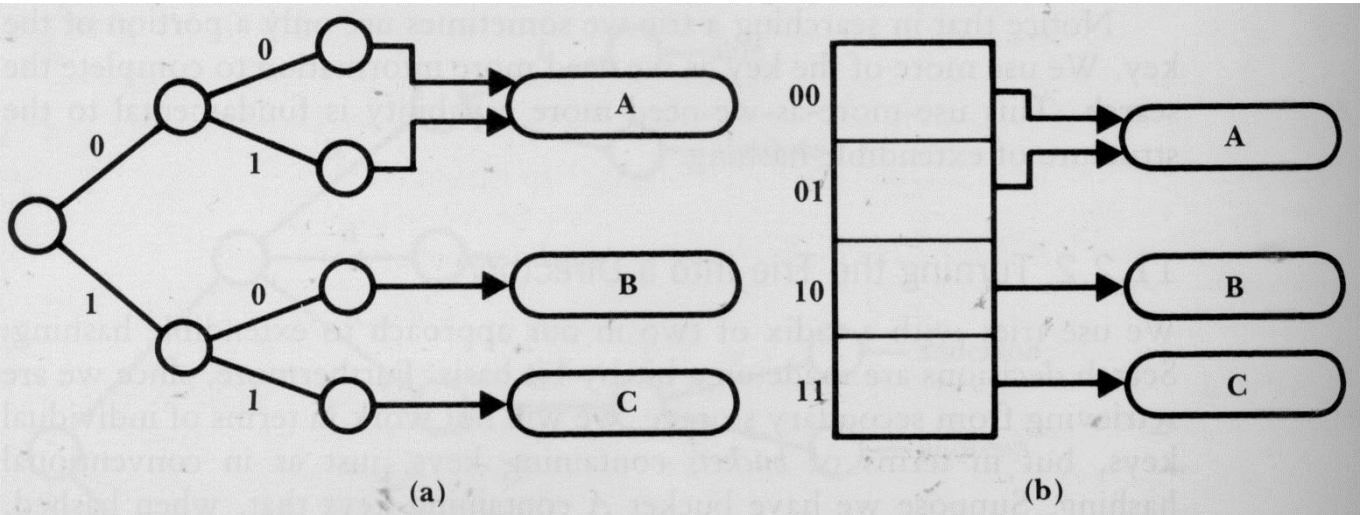
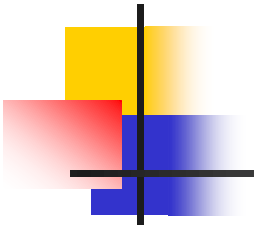


FIGURE 11.4 The trie from Fig. 11.3 transformed first into a complete binary tree, and then flattened into a directory to the buckets.

Transformando a trie em um diretório



- Uma vez "completa", trie pode ser representada por um vetor
- O vetor fornece acesso direto aos endereços
- Exemplo: endereço começando com os bits 10 pode ser encontrado a partir de um ponteiro na posição 10_2 do diretório

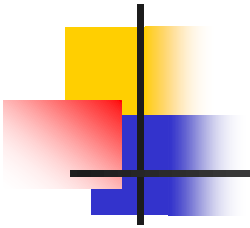
Estruturas de dados do Diretório e Buckets

Record Type: Bucket

Depth	integer count of the number of bits usede “in common” by the keys in this bucket
Count	integer count of the number of keys in the bucket
Key [ ]	array [1..Max_Bucket_Size] of strings to hold keys

Record Type: Directory_Cell

Bucket_Ref	relative record number (RRN) or other reference to a specific **Bucket** record on disk

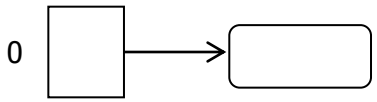


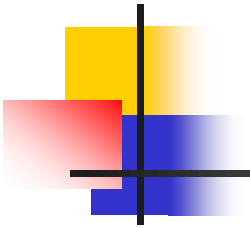
Profundidade do Diretório

- Profundidade do cesto: indicação do número de bits da chave necessários para determinar quais registros ele contém
  - Quanto mais ponteiros do diretório para o cesto, menor sua profundidade
  - Essa informação pode ser mantida junto ao cesto
- No exemplo anterior:
  - Cesto A: profundidade 1
  - Cestos B e C: profundidade 2

Profundidade do Diretório

- Maior profundidade dos cestos, ou número de bits necessários para distinguir os cestos, ou número de bits dos endereços (binários) do diretório, ou $\log_2 E$, onde E é o tamanho do diretório
- No exemplo: Profundidade do diretório = 2
- Inicialmente, a profundidade do diretório é 0 e há um único bucket





Splitting para tratar overflow

- Se um registro precisa ser inserido e não há espaço no cesto associado ao seu endereço base, então o cesto é sub-dividido (*spliting*). Isso é feito adicionando mais um bit aos endereços
  - Distribui-se o conjunto de chaves do bucket cheio de modo que o próximo bit do endereço seja distinto nos 2 buckets
  - Ver exemplo anterior, buckets B (10) e C (11)
- se o novo espaço de endereçamento já estava previsto no diretório, nenhuma alteração é necessária
- senão, é necessário dobrar o espaço de endereçamento do diretório para acomodar o novo bit

Subdivisão para tratar overflow

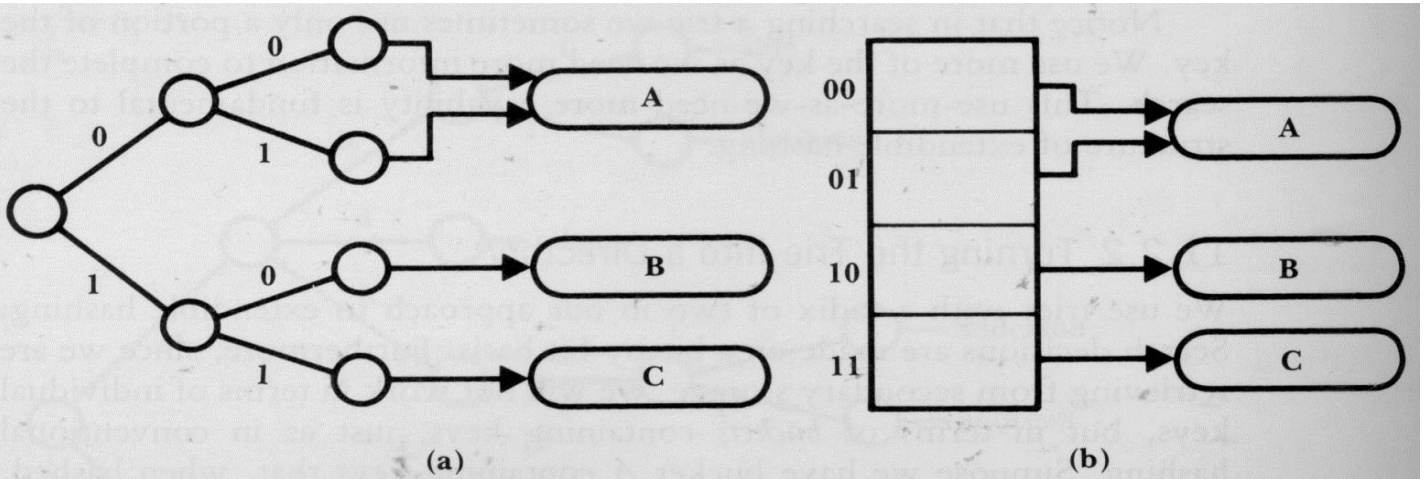


FIGURE 11.4 The trie from Fig. 11.3 transformed first into a complete binary tree, and then flattened into a directory to the buckets.

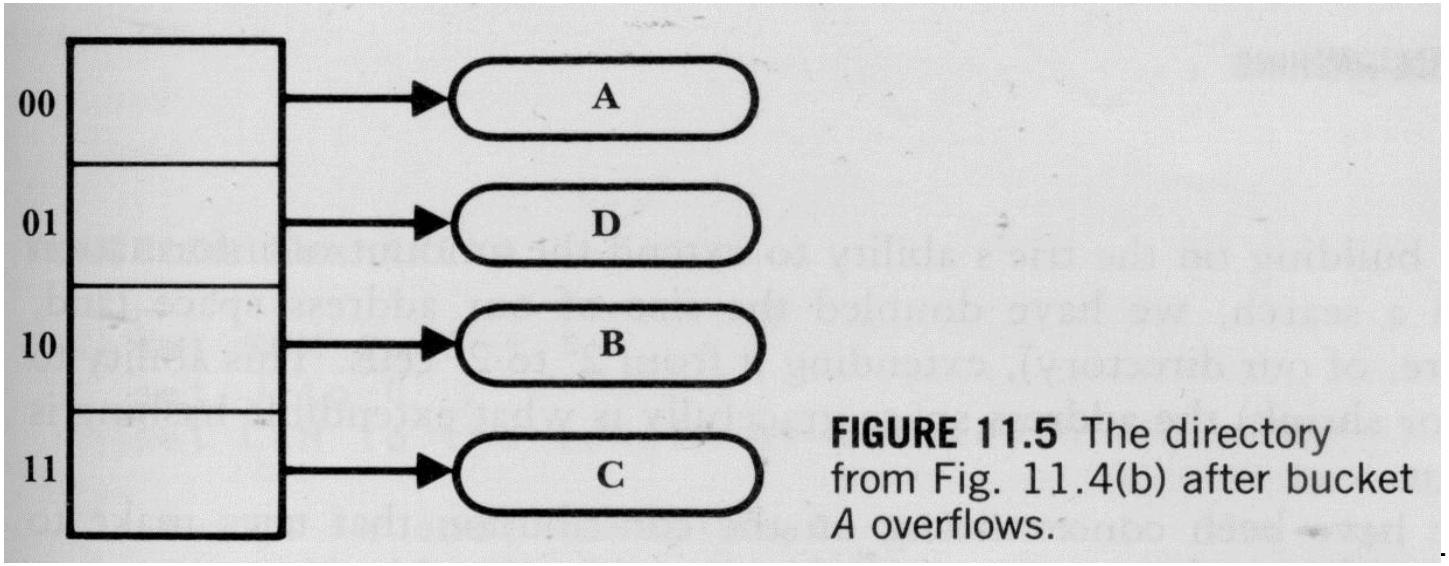


FIGURE 11.5 The directory from Fig. 11.4(b) after bucket *A* overflows.

Subdivisão para tratar overflow

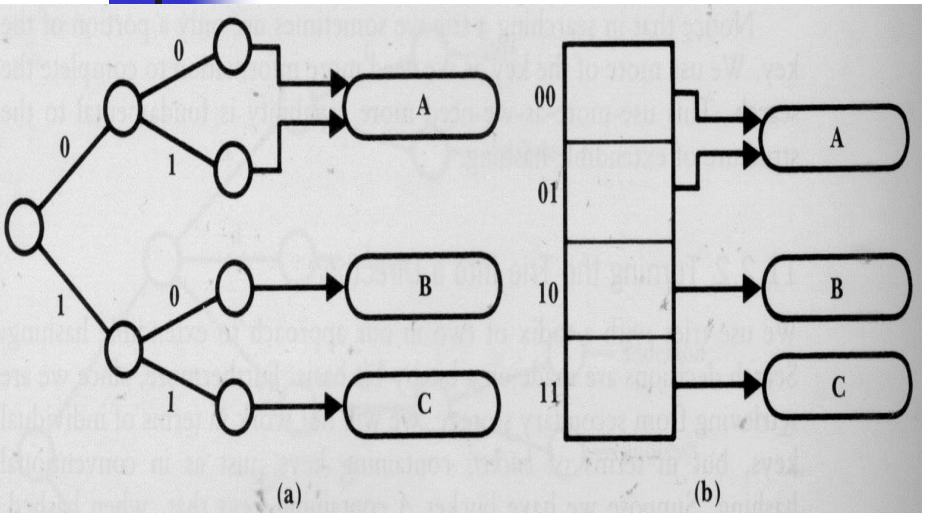
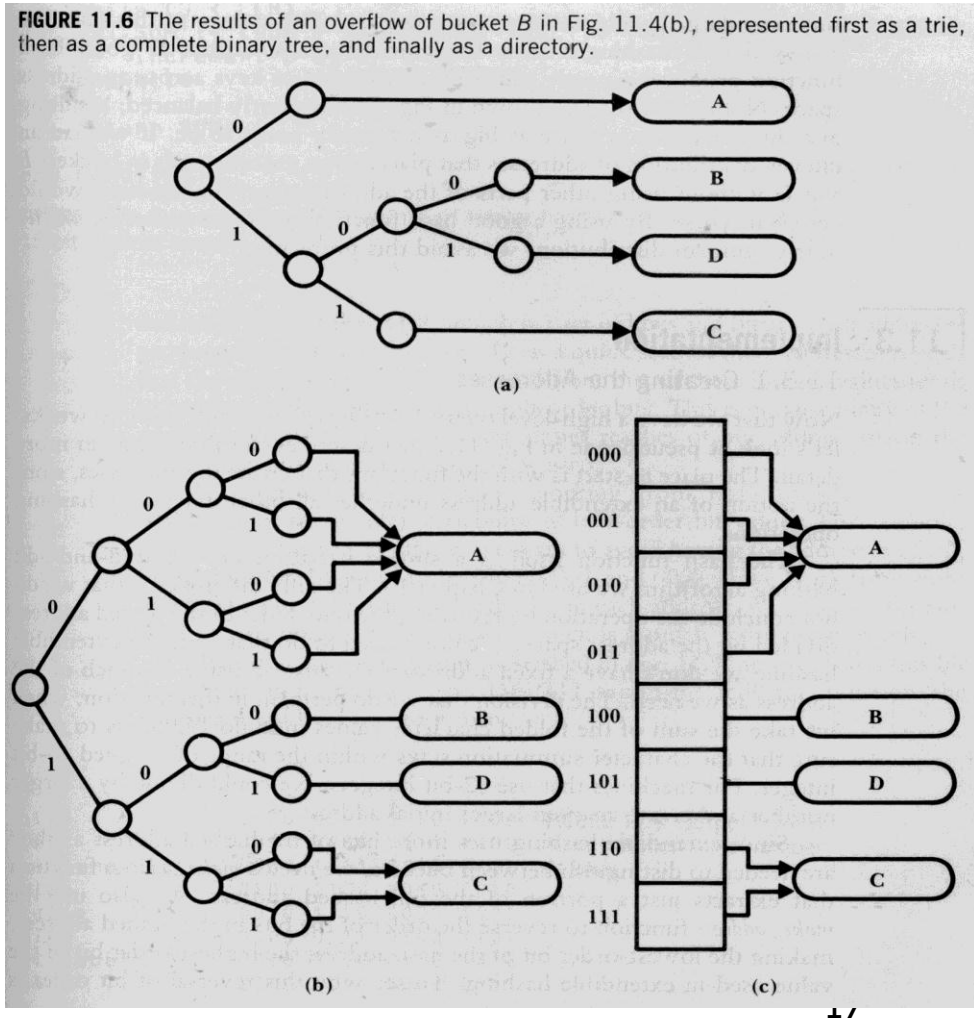
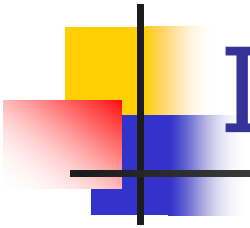


FIGURE 11.4 The trie from Fig. 11.3 transformed first into a complete binary tree, and then flattened into a directory to the buckets.



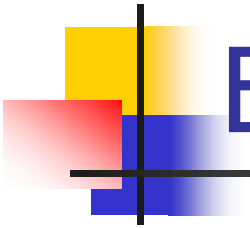


Inserção de chave

- Seja *d* a profundidade do índice, dada pela maior profundidade dos cestos
- Localiza chave no diretório: seja *i* a profundidade do seu cesto
- Se a inserção da chave provoca a sub-divisão do cesto, existem 2 casos a considerar: i < d e i = d

Inserção de chave

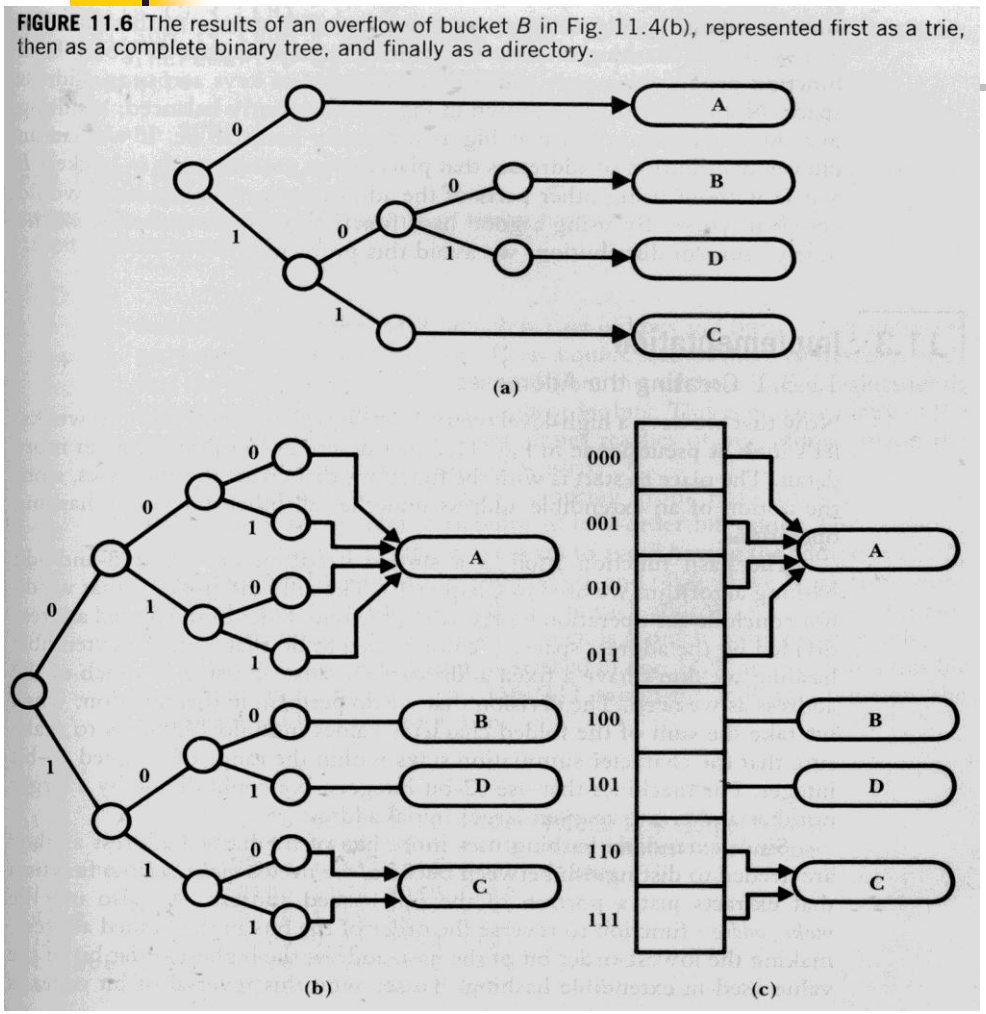
- Se **i < d** (Ex. Bucket A: há endereço disponível no diretório para um novo bucket)
  - Remaneja os registros entre os 2 cestos
  - Insere nova chave no cesto adequado
  - Altera a profundidade de ambos os cestos
- Se **i = d** (Ex. Bucket B: já usa todos os bits possíveis)
  - É necessário dobrar o tamanho do diretório
  - Profundidade do índice passa a ser d + 1, assim como a dos cestos envolvidos na inserção
  - Distribui-se o conjunto de chaves do bucket cheio de modo que o próximo bit do endereço seja distinto nos 2 buckets
  - Antigo conteúdo de todas as demais posições do índice copiado para o novo índice



Eliminação de chave

- Localiza chave no diretório
- Se encontrada, elimina a chave do seu cesto
- Verifica-se se o cesto possui um "buddy bucket"
  - Um par de cestos "buddy" é formado por dois cestos que são descendentes imediatos do mesmo nó na trie
- Se o "buddy bucket" existe, então verifica se é possível unir os cestos "buddy"
- Verifica se é possível diminuir (colapsar) o tamanho do diretório

Eliminação - Exemplo



Suponha:

(a) uma eliminação em B;

(b) D tem poucos elementos;

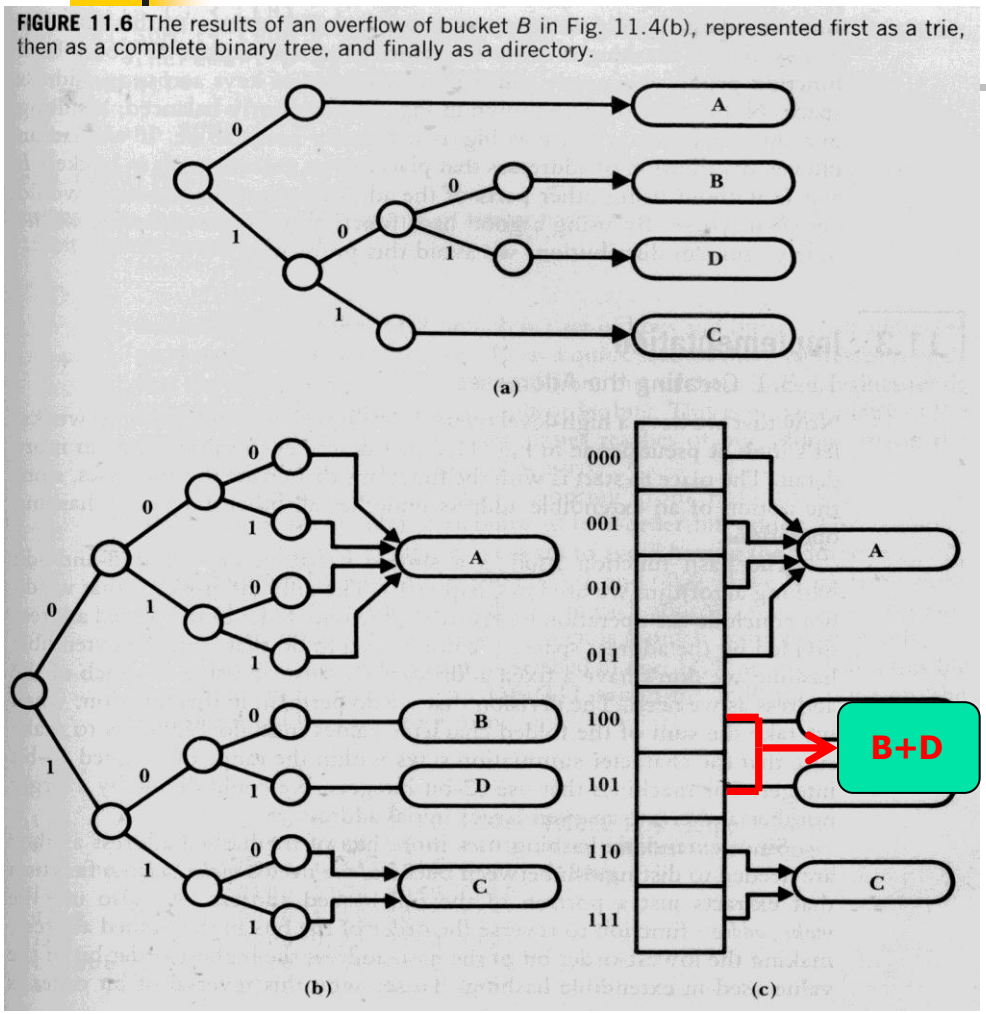
(c) após a eliminação, os elementos de B+D cabem num único bucket

Então:

é possível **unir** B com D

prof. A=1; prof. B e D=3; prof. C=2; prof. Diretório=3

Eliminação - Exemplo



Suponha:

(a) uma eliminação em B;

(b) D tem poucos elementos;

(c) após a eliminação, os elementos de B+D cabem num único bucket

Então:

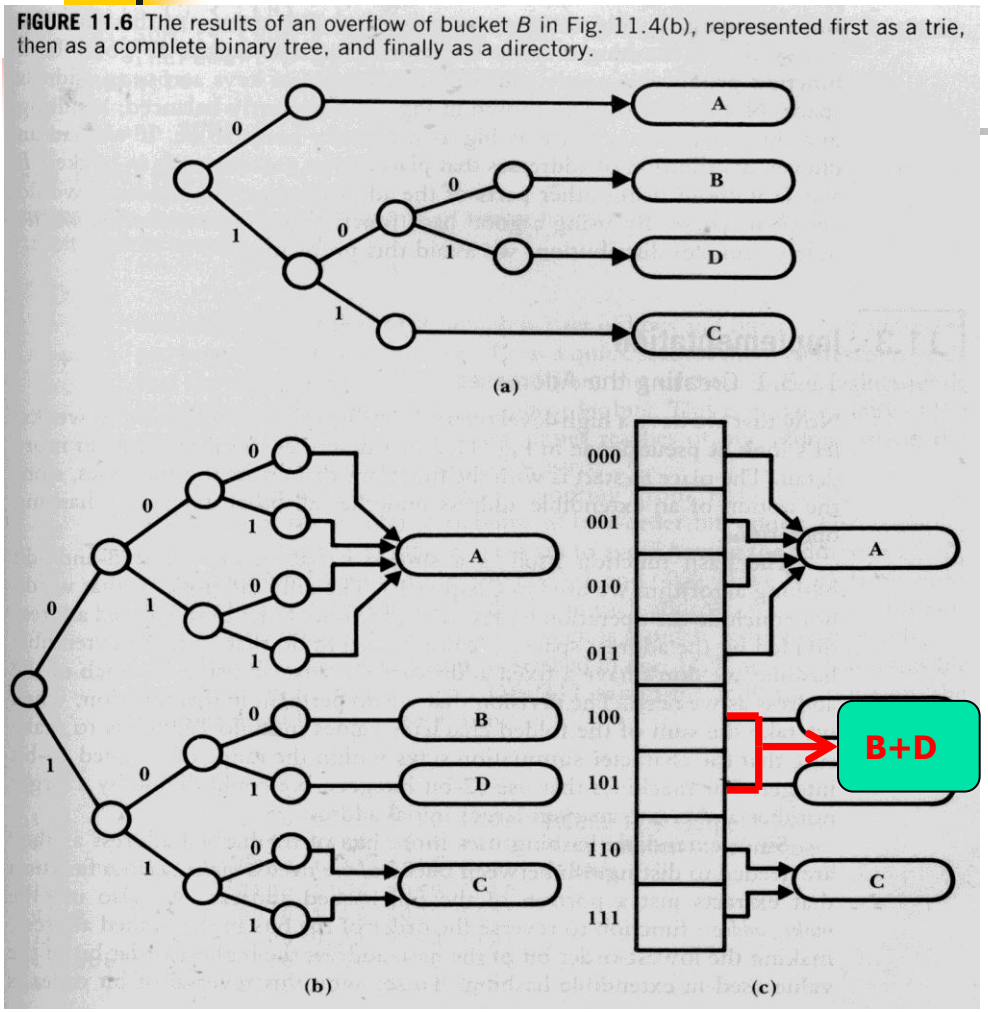
é possível **unir** B com D

Agora todos diretórios têm profundidade menor que o diretório, então

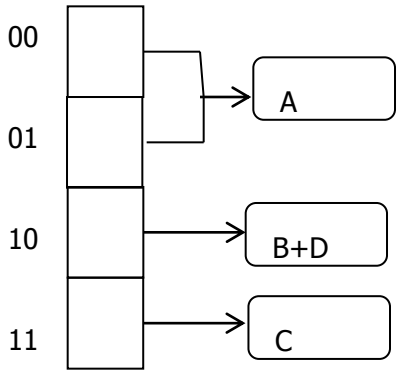
É possível **colapsar** o diretório

prof. A=1; prof. B e D=3; prof. C=2; prof. Diretório=3

No exemplo:



É possível colapsar o diretório, reduzindo-o à metade:

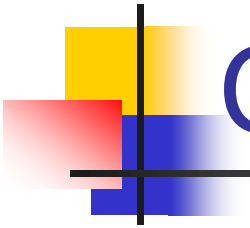


E atualizar as profundidades:
A=2;B+D=1;C=1;Diretório=2

prof. A=1; prof. (B+D)=2; prof. C=2; prof. Diretório=3

Par de cestos “buddy”

- Se a profundidade do cesto for menor que a profundidade do diretório, tal cesto não tem “buddy”
- Caso o “buddy” exista, pode-se determinar o endereço do cesto “buddy” usando o do cesto atual
  - No exemplo:
    - Endereço de B: 10**0**; Endereço de C: 10**1**

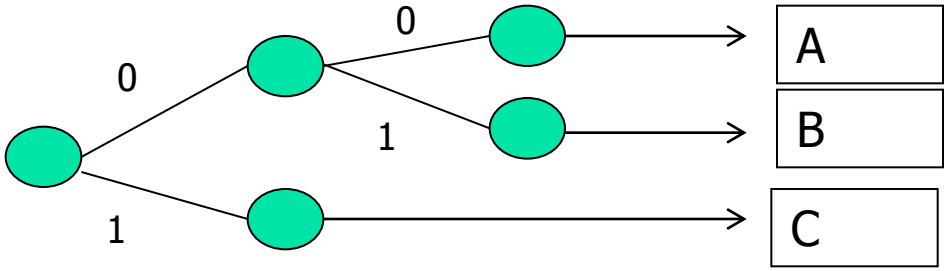


Colapso de diretórios

- Se um par de cestos “buddy” é unido, pode acontecer que todo cesto tenha, no mínimo, um par de endereços referenciando-o
  - Verifica-se se todas as profundidades dos cestos são menores que a do diretório
- Neste caso, o diretório pode ser colapsado, e seu tamanho reduzido pela metade

Exercícios

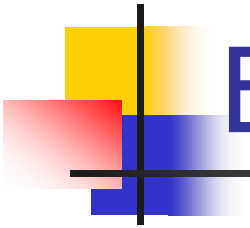
(1) Considere a seguinte *trie* de ordem (raio) 2, com ponteiros para *buckets* com capacidade para abrigar 100 chaves (ou registros):



a. Desenhe a *trie* estendida e o diretório de endereços hash correspondente.

b. Considerando que os buckets A, B e C contém, respectivamente, 100, 50 e 03 registros, dê a configuração do diretório, e a condição de cada bucket após a inserção de uma nova chave cujo valor da função hash é 00.

c. Ainda na configuração inicial, considere agora que todas as chaves de B são eliminadas. O que acontece com o diretório?



Exercícios

(2) Considere um máximo de 3 elementos por bucket, e que a função hash gera 4 bits para uma chave. Simule a inserção de chaves que geram os seguintes endereços:

0000, 1000, 1001, 1010, 1100, 0001, 0100, 1111, 1011